

Файлы-потоки

Механизм ввода-вывода, разработанный для обычного языка C, не соответствует общепринятому сегодня стилю объектно-ориентированного программирования, кроме того, он активно использует операции с указателями, считающиеся потенциально небезопасными в современных защищённых средах выполнения кода. Альтернативой при разработке прикладных приложений является механизм стандартных классов ввода-вывода, предоставляемый стандартом языка C++.

Открытие файлов

Наиболее часто применяются классы `ifstream` для чтения, `ofstream` для записи и `fstream` для модификации файлов.

Все поточные классы ввода-вывода являются косвенными производными от общего предка `ios`, полностью наследуя его функциональность. Так, режим открытия файлов задает член данных перечисляемого типа `open_mode`, который определяется следующим образом:

```
enum open_mode { app, binary, in, out, trunc, ate };
```

Ниже приведены возможные значения флагов и их назначение.

Режим	Назначение
<code>in</code>	Открыть для ввода (выбирается по умолчанию для <code>ifstream</code>)
<code>out</code>	Открыть для вывода (выбирается по умолчанию для <code>ofstream</code>)
<code>binary</code>	Открыть файл в бинарном виде
<code>app</code>	Присоединять данные; запись в конец файла
<code>ate</code>	Установить файловый указатель на конец файла
<code>trunc</code>	Уничтожить содержимое, если файл существует (выбирается по умолчанию, если флаг <code>out</code> указан, а флаги <code>ate</code> и <code>app</code> — нет)

Например, чтобы открыть файл с именем `test.txt` для чтения данных в бинарном виде, следует написать:

```
ifstream file;
```

```
file.open ("test.txt", ios::in | ios::binary);
```

Оператор логического ИЛИ (`|`) позволяет составить режим с любым сочетанием флагов. Так, чтобы, открывая файл по записи, случайно не затереть существующий файл с тем же именем, надо использовать следующую форму:

```
ofstream file;
```

```
file.open ("test.txt", ios::out | ios::app);
```

Предполагается, что к проекту подключён соответствующий заголовочный файл:

```
#include <fstream.h>
```

Для проверки того удалось ли открыть файл, можно применять конструкцию

```
if (!file) {
```

```
//Обработка ошибки открытия файла
```

```
}
```

Операторы включения и извлечения

Переопределённый в классах работы с файлами оператор включения (<<) записывает данные в файловый поток. Как только вы открыли файл для записи, можно записывать в него текстовую строку целиком:

```
file << "Это строка текста";
```

Можно также записывать текстовую строку по частям:

```
file << "Это " << "строка " << "текста";
```

Оператор endl завершает ввод строки символом "возврат каретки":

```
file << "Это строка текста" << endl;
```

С помощью оператора включения несложно записывать в файл значения переменных или элементов массива:

```
ofstream file ("Temp.txt");  
char buff[] = "Текстовый массив содержит переменные";  
int vx = 100;  
float pi = 3.14159;  
file << buff << endl << vx << endl << pi << endl;
```

В результате выполнения кода образуется три строки текстового файла Temp.txt:

Текстовый массив содержит переменные

100

3.14159

Обратите внимание, что числовые значения записываются в файл в виде текстовых строк, а не двоичных значений.

Оператор извлечения (>>) производит обратные действия. Казалось бы, чтобы извлечь символы из файла Temp.txt, записанного ранее, нужно написать код наподобие следующего:

```
ifstream file ("Temp.txt");  
char buff[100];  
int vx;  
float pi;  
file >> buff >> vx >> pi;
```

Однако оператор извлечения остановится на первом попавшемся разделителе (символе пробела, табуляции или новой строки). Таким образом, при разборе предложения "Текстовый массив содержит переменные" только слово "Текстовый" запишется в массив buff, пробел игнорируется, а слово "массив" станет значением целой переменной vx и исполнение кода "пойдет вразнос" с неминуемым нарушением структуры данных. Далее, при обсуждении класса ifstream, будет показано, как правильно организовать чтение файла из предыдущего примера.

Класс ifstream: чтение файлов

Как следует из расшифровки названия, класс ifstream предназначен для ввода файлового потока. Далее перечислены основные методы класса. Большая часть из них унаследована от класса istream и перегружена с расширением родительской функциональности. К примеру, функция get, в зависимости от параметра вызова, способна считывать не только одиночный символ, но и символьный блок.

Метод	Описание
-------	----------

open	Открывает файл для чтения
get	Читает один или более символов из файла
getline	Читает символьную строку из текстового файла или данные из бинарного файла до определенного ограничителя
read	Считывает заданное число байт из файла в память
eof	Возвращает ненулевое значение (true), когда указатель потока достигает конца файла
peek	Выдает очередной символ потока, но не выбирает его
seekg	Перемещает указатель позиционирования файла в заданное положение
tellg	Возвращает текущее значение указателя позиционирования файла
close	Закрывает файл

Теперь понятно, как нужно модифицировать предыдущий пример, чтобы использование оператора извлечения данных давало ожидаемый результат:

```
ifstream file("Temp.txt");
char buff[100];
int vx;
float pi;
file.getline(buff, sizeof(buff));
file >> vx >> pi;
```

Метод getline прочитает первую строку файла до конца, а оператор >> присвоит значения переменным.

Следующий пример показывает добавление данных в текстовый файл с последующим чтением всего файла. Цикл while (1) используется вместо while(!file2.eof()).

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream file;
    file.open("test.txt", ios::out/ios::app);
    if (!file) {
        cout << "File error - can't open to write data!";
        cin.sync(); cin.get(); return 1;
    }
    for (int i=0; i<10; i++) file << i << endl;
    file.close();
    ifstream file2;
    file2.open("test.txt", ios::in);
    if (!file2) {
        cout << "File error - can't open to read data!";
        cin.sync(); cin.get(); return 2;
    }
    int a,k=0;
    while (1) {
        file2 >> a;
```

```

    if (file2.eof()) break;
    cout << a << " ";
    k++;
}
cout << endl << "K=" << k << endl;
file2.close();
cin.sync(); cin.get(); return 0;
}

```

В следующем примере показан цикл считывания строк из файла test.txt и их отображения на консоли.

```

#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream file;          // создать поточный объект file
    file.open("test.txt");  // открыть файл на чтение
    if (!file) return 1;    // возврат по ошибке открытия
    char str[80];           // статический буфер строки
    // Считывать и отображать строки в цикле, пока не eof
    while (!file.getline(str, sizeof(str)).eof())
        cout << str << endl; // вывод прочитанной строки на экран
    cin.sync(); cin.get(); return 0;
}

```

Этот код под ОС Windows также зависит от наличия в последней строке файла символа перевода строки, надежнее было бы сделать так:

```

while (1) {
    if (file.eof()) break;
    file.getline(str, sizeof(str));
    cout << str << endl;
}

```

Явные вызовы методов open и close не обязательны. Действительно, вызов конструктора с аргументом позволяет сразу же, в момент создания поточного объекта file, открыть файл:

```
ifstream file("test.txt");
```

Вместо метода close можно использовать оператор delete, который автоматически вызовет деструктор объекта file и закроет файл. Код цикла while обеспечивает надлежащую проверку признака конца файла.

Класс ofstream: запись файлов

Класс ofstream предназначен для вывода данных из файлового потока. Далее перечислены основные методы данного класса.

Метод	Описание
open	Открывает файл для записи
put	Записывает одиночный символ в файл
write	Записывает заданное число байт из памяти в файл

seekp	Перемещает указатель позиционирования в указанное положение
tellp	Возвращает текущее значение указателя позиционирования файла
close	Закрывает файл

Описанный ранее оператор включения удобен для организации записи в текстовый файл:

```
ofstream file ("temp.txt");
if (!file) return;
for (int i=1; i<=3; i++)
file << "Строка " << i << endl;
file.close();
```

Бинарные файлы

В принципе, бинарные данные обслуживаются наподобие текстовых. Отличие состоит в том, что если бинарные данные записываются в определенной логической структуре, то они должны считываться из файла в переменную того же структурного типа.

Первый параметр методов write и read (адрес блока записи/чтения) должен иметь тип символьного указателя char *, поэтому необходимо произвести явное преобразование типа адреса структуры void *. Второй параметр указывает, что бинарные блоки файла имеют постоянный размер байтов независимо от фактической длины записи. Следующее приложение дает пример создания и отображения данных простейшей записной книжки. Затем записи файла последовательно считываются и отображаются на консоли.

```
#include <iostream>
#include <fstream>
#include <locale>
using namespace std;

struct Notes { // структура данных записной книжки
char Name[60]; // Ф.И.О.
char Phone[16]; // телефон
int Age; // возраст
};

int main() {
setlocale(LC_ALL, "Russian");
Notes Note1= { "Грозный Иоанн Васильевич", "не установлен", 60 };
Notes Note2= { "Годунов Борис Федорович ", "095-111-2233 ", 30 };
Notes Note3= { "Романов Петр Михайлович ", "812-333-2211 ", 20 };
ofstream ofile("Notebook.dat", ios::binary);
ofile.write((char*)&Note1, sizeof(Notes)); // 1-й блок
ofile.write((char*)&Note2, sizeof(Notes)); // 2-й блок
ofile.write((char*)&Note3, sizeof(Notes)); // 3-й блок
ofile.close(); // закрыть записанный файл
ifstream ifile("Notebook.dat", ios::binary);
Notes Note; // структурированная переменная
char str[80]; // статический буфер строки
// Считывать и отображать строки в цикле, пока не eof
```

```

while (!ifile.read((char*)&Note, sizeof(Notes)).eof()) {
    sprintf(str, "%s\tТел: %s\tВозраст: %d",
        Note.Name, Note.Phone, Note.Age);
    cout << str << endl;
}
ifile.close();          // закрыть прочитанный файл
cin.sync(); cin.get(); return 0;
}

```

В результате выполнения этого кода образуется бинарный файл Notebook.dat из трех блоков размером по 80 байт каждый (при условии, что символы - однобайтовые). Естественно, вы можете использовать другие поточные методы и проделывать любые операции над полями определенной структуры данных.

Класс fstream: произвольный доступ к файлу

Предположим что в нашей записной книжке накопилось 100 записей, а мы хотим считать 50-ю. Конечно, можно организовать цикл и прочесть все записи с первой по заданную. Очевидно, что более целенаправленное решение - установить указатель позиционирования файла pos прямо на запись 50 и считать ее:

```

ifstream ifile("Notebook.dat", ios::binary);
int pos = 49 * sizeof(Notes);
ifile.seekg(pos); // поиск 50-й записи
Notes Note;
//Notes – описанная выше структура "запись"
ifile.read((char*)&Note, sizeof(Notes));

```

Подобные операции поиска эффективны, если файл состоит из записей известного и постоянного размера. Чтобы заменить содержимое произвольной записи, надо открыть поток вывода в режиме модификации:

```

ofstream ofile ("Notebook.dat",
    ios::binary | ios::ate);
int pos = 49 * sizeof(Notes);
ofile seekp(pos); // поиск 50-й записи
Notes Note50 =
    {"Ельцин Борис Николаевич", "095-222-3322", 64};
ofile.write((char*)&Note, sizeof(Notes)); // замена

```

Если не указать флаг ios::ate (или ios::app), то при открытии бинарного файла Notebook.dat его предыдущее содержимое будет стерто!

Наконец, можно открыть файл одновременно для чтения/записи, используя методы, унаследованные поточным классом fstream от своих предшественников. Поскольку класс fstream произведен от istream и ostream (родителей ifstream и ofstream соответственно), все упомянутые ранее методы становятся доступными в приложении.

В следующем примере показана перестановка первой и третьей записей файла Notebook.dat.

```

#include <iostream>
#include <fstream>
#include <locale>
using namespace std;

```

```

struct Notes { char Name[60]; char Phone[16]; int Age; };

```

```

int main() {
    setlocale(LC_ALL, "Russian");
    Notes Note1, Note3;
    // Открыть файл на чтение/запись одновременно
    fstream file("Notebook.dat", ios::binary | ios::in | ios::out);
    file.seekg(2 * sizeof(Notes)); // найти и считать Note3
    file.read((char*)&Note3, sizeof(Notes));
    file.seekg(0); // найти и считать Note1
    file.read((char*)&Note1, sizeof(Notes));
    file.seekg(0); // Note1 <= Note3
    file.write((char*)&Note3, sizeof(Notes));
    file.seekg(2 * sizeof(Notes)); // Note3 <= Note1
    file.write((char*)&Note1, sizeof(Notes));
    char str[80];
    // Считывать и отображать записи в цикле, пока не eof
    file.seekg(0); // вернуться к началу файла
    while (!file.read((char*)&Note1, sizeof(Notes)).eof()) {
        sprintf(str, "%s\tТел: %s\tВозраст: %d",
            Note1.Name, Note1.Phone, Note1.Age);
        cout << str << endl;
    }
    file.close();
    cin.sync(); cin.get(); return 0;
}

```

В конструкторе объекта file надо указать флаги ios::in и ios::out, разрешая одновременное выполнение операций чтения и записи. В результате выполнения этого кода первая и третья записи бинарного файла Notebook.dat поменяются местами.